

CS545: Distributed Computing with R Using Snowfall

Chuck Anderson

Department of Computer Science
Colorado State University

Fall, 2009

Outline

CS545:
Distributed
Computing with R
Using Snowfall

Chuck Anderson

Snowfall

Installation

Initialize a Cluster

Snowfall

Installation

Initialize a Cluster

Install Packages for Snowfall

- Check out the `HighPerformanceComputing` link at <http://cran.r-project.org/web/views>

Install Packages for Snowfall

- Check out the `HighPerformanceComputing` link at <http://cran.r-project.org/web/views>
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall

Install Packages for Snowfall

- Check out the HighPerformanceComputing link at <http://cran.r-project.org/web/views>
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
- You need snow and snowfall

```
install.packages(c("snow", "snowfall"))
```

Install Packages for Snowfall

- Check out the HighPerformanceComputing link at <http://cran.r-project.org/web/views>
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
- You need snow and snowfall

```
install.packages(c("snow", "snowfall"))
```

- We will set up a cluster that communicates with TCP/IP sockets, because this works on Linux and MS Windows without installing any additional software.

Initialize a Cluster

- To initialize a cluster

```
sfnit ( parallel = TRUE, cpus = 4, type = "SOCK")
```

`parallel` may be set to `FALSE` to run on a single

Initialize a Cluster

- To initialize a cluster

```
sfnit ( parallel = TRUE, cpus = 4, type = "SOCK")
```

`parallel` may be set to `FALSE` to run on a single

- You may also specify which machines to use

```
sfnit ( parallel = TRUE, type="SOCK",  
       socketHosts=c( "corn", "cucumber", "cucumber", "radish"))
```

Without the `socketHosts` argument, you will be running on just your local host.

Initialize a Cluster

- To initialize a cluster

```
sflnit ( parallel = TRUE, cpus = 4, type = "SOCK")
```

`parallel` may be set to `FALSE` to run on a single

- You may also specify which machines to use

```
sflnit ( parallel = TRUE, type="SOCK",  
        socketHosts=c( "corn", "cucumber", "cucumber", "radish"))
```

Without the `socketHosts` argument, you will be running on just your local host.

- Or, you may just call

```
sflnit ()
```

in your code and set the argument values in the R command line

```
R --no-save ---no-restore --args --parallel --cpus=4 \  
    --type=SOCK
```

The items that follow `--args` are parsed by `sfInit` using the R function `commandArgs`.

Which Hosts?

- To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that

Which Hosts?

- To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that
 - consults a file of host names and maximum number of CPUs to use for each,

Which Hosts?

- To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that
 - consults a file of host names and maximum number of CPUs to use for each,
 - uses the unix command `rup` to determine the current load on each host,

Which Hosts?

- To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that
 - consults a file of host names and maximum number of CPUs to use for each,
 - uses the unix command `rup` to determine the current load on each host,
 - calculates the number of CPUs to use as the given maximum minus the current load,

Which Hosts?

- To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that
 - consults a file of host names and maximum number of CPUs to use for each,
 - uses the unix command `rup` to determine the current load on each host,
 - calculates the number of CPUs to use as the given maximum minus the current load,
 - duplicates the host name that many times, and returns the list of all host names.

Using snowfallSelectHosts

- Here is a text file of machine names and maximum numbers of CPUs:

```
brussels-sprout 5  
cauliflower 5  
horseradish 5  
kelp 5  
romanesco 5
```

These machines have 8 cores.

Using snowfallSelectHosts

- Here is a text file of machine names and maximum numbers of CPUs:

```
brussels-sprout 5  
cauliflower 5  
horseradish 5  
kelp 5  
romanesco 5
```

These machines have 8 cores.

- If this file is names machines, then here is how I would use snowfallSelectHosts

```
hosts <- snowfallSelectHosts("machines",localhost=TRUE,print=FALSE)  
sflnit ( parallel = TRUE, type = "SOCK", socketHosts = hosts)  
snowfall 1.70 initialized : parallel execution on 17 CPUs.
```

Using snowfallSelectHosts

- You can also see the processing of each host.
Remember to stop the cluster we just created first.

Using snowfallSelectHosts

- You can also see the processing of each host.
Remember to stop the cluster we just created first.

```
sfStop()  
hosts <- snowfallSelectHosts("machines",localhost=TRUE,print=TRUE) ## Changed la
```

which produces this output

```
Read 5 hosts from file "machines"  
Using 0 of 5 slot(s) on brussels-sprout  
Using 0 of 5 slot(s) on cauliflower  
Using 0 of 5 slot(s) on horseradish  
Using 3 of 5 slot(s) on kelp  
Using 0 of 5 slot(s) on romanesco  
Using 7 of 8 slot(s) on thoumire  
for total of 10 slots
```

after which you continue with

```
sflnit ( parallel = TRUE, type = "SOCK", socketHosts = hosts)  
snowfall 1.70 initialized : parallel execution on 10 CPUs.
```

Using new distributed apply functions

- Say we want to square each value of a list named data.
Can use `sfLapply`.

```
data <- 1:5
result <- sfLapply(data, function(x) {x * x} )
print ( result )
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25
```

Using new distributed apply functions

- Can use the automatic load balancing provided by snowfall by using `sfClusterApplyLB`.

```
data <- 1:5
result <- sfClusterApplyLB(data, function(x) {x * x} )
print ( result )
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25
```

- Must also make sure each process loads the needed libraries. Do so with `sfLibrary`.

- Must also make sure each process loads the needed libraries. Do so with `sfLibrary`.
- Now a bigger example.

```

sfNnit ( parallel = TRUE, cpus = 4, type="SOCK", socketHosts=hosts)
sfLibrary (randomForest)
data( iris )
boot.n <- 10
boot.index <- t(sapply(1:boot.n,function(b){sample(1:nrow( iris ), replace=TRUE)
not.in.sample <- list()
for ( i in 1:boot.n)
  not.in.sample[[i]] <- (1:nrow(iris))[-unique(boot.index[i ,])]
boot.fun <- function(actual.sample) {
  sample.fit <- randomForest(Species ~ ., data=iris[boot.index[ actual.sample ],])
  prediction <- predict(sample.fit, newdata=iris[not.in.sample[[actual.sample ]],1:4])
  sum(prediction != iris [not.in.sample[[actual.sample ]],5]) /length( prediction)
}
sfExport(" iris ", "boot.index", "not.in.sample")
res <- sfClusterApplyLB(1:boot.n, boot.fun)
error <- mean(unlist(res))
print ( error )

```

Running it shows

```

snowfall 1.70 initialized : parallel execution on 7 CPUs.
Library randomForest loaded.
Library randomForest loaded in cluster .
[1] 0.0492253

```